AMENDMENTS TO THE SPECIFICATION

The disclosure was objected to due to informalities. In particular, appropriate correction of capitalization was required for several instances of terms utilizing the trademarked word "Java™". The amendments to the specification recited below address and eliminate these informalities as well as other informalities noted during review of the Specification.

Please replace the paragraph that begins on line 13 of page 3, with the following amended paragraph: KB6-17-10

While interaction between emulation language code and native language code can be relatively simple, some aspects are absolutely critical, ~~inparticular~~ in particular the coordinated passage of information between emulation language code and native language code. A Java Native Language Interface (JNI) is one attempt at providing a protocol for objects in a Java emulation language to interact with a native language (e.g., C, C++, etc.). To beneficially participate in information processing, it is typically desirable for a JNI to permit a native method to interact to some extent with the internal state of a Java ~~virtual machine~~ Virtual Machine (JVM) instance, including pass and return data, access instance and class variables, invoke instance and class methods, access arrays, etc. It is also desirable for a JNI to support portability in these activities and a JNI typically attempts to achieve portability through the use of pointers to memory locations of other pointers, variables and/or functions.

Please replace the paragraph that begins on line 1 of page 8, with the following amended paragraph: KB6-M-10

In step 11, an emulation language virtual machine (e.g. a Java ~~virtual machine~~ Virtual Machine (JVM)) is initialized. An emulation language virtual machine creates a runtime environment. For example, a runtime environment can include a class loader subsystem and an execution engine. The behavior of an initialized emulation language virtual machine instance can be defined in terms of subsystems, memory areas, data types and instructions.

Please replace the paragraph that begins on line ~~23~~ 10 of page ~~17~~ 12, with the following amended paragraph: KB6-17-10

Figure 3 is a block diagram representation of an emulation and native language interface test architecture in accordance with one embodiment of the present invention. In one exemplary implementation, the emulation and native language interface test architecture emulates a Java compatible architecture (e.g., compatible with Java ~~virtual machine~~ Virtual Machine specification, Java language compatible, Java bytecode compatible, etc.). In one embodiment of the present invention, instructions for causing a computer system to implement an emulation and native language interface test architecture (e.g., JNI interface test architecture) are stored and embodied on a computer usable storage medium (e.g., as computer readable code).

Please replace the paragraph that begins on line ~~17~~ 9 of page ~~13~~ 14, with the following amended paragraph: KB6-17-10

Emulation class loading module 331 places emulation class information into memory (e.g., runtime data area 333 ~~area 233~~) for processing. In one embodiment of the present invention, emulation class loading module 331 ~~module 231~~ is Java compatible. In one embodiment of the present invention, emulation class information includes Java classes. Java classes are code segments defining objects which are instances of a class (e.g., "component" characteristics or features included in the class). The Java class definitions can include fields and methods. The fields or class variables ("variables") can include data (e.g., integers or characters) which can be private data accessible by a single class or public data accessible by a plurality of classes. The data can also be characterized as static data associated with class objects as a whole (e.g., common to each instance of the class) or dynamic data associated with class objects individually. The methods perform tasks or functions (e.g., a subroutine). The methods can also call other methods via invocations and/or can pass data to other objects.

3 of 3 10/628960 KB 6-17-10

22 15 KB 6-17-10

Please replace the paragraph that begins on line 22 of page 14, with the following amended paragraph:

Figure 4 is a block diagram of JNI testing system 400, one embodiment of a computer system on which the present invention can be implemented. For example, computer system 400 can be utilized to implement Java Native Language Interface (JNI) test method 100 and emulation and native language interface test system 300. JNI testing system 400 includes communication bus 457, processor 451, memory 452, input component 453, bulk storage component 454 (e.g., a disk drive), network communication port 459 and display module 455. Communication bus 457 is coupled to central processor 451, memory 452, input component 453, bulk storage component 454, network communication port 459 and display module 455. The components of [[JIN]] JNI testing system 400 cooperatively function to provide a variety of functions, including performing a "machine" emulation with JNI testing capabilities in accordance with a present invention. Communication bus 407 communicates information. Processor 451 processes information and instructions, including instructions for testing a JNI. In one embodiment of the present invention, processor 451 performs a JNI testing process (e.g., method 10, 100, 200, etc). Memory 452 stores information and instructions, including instructions for testing the JNI. Bulk storage component 454 also provides storage of information. Input component 453 facilitates communication of information to computer system 450. Display module 455 displays information to a user. Network communication port 459 provides a communication port for communicatively coupling with a network.